자료구조 Homework #5

충북대학교 컴퓨터공학과 2022040014 오재식

CircularQ.c

```c
/* circularQ.c
 *
 *  Data Structures, Homework #5
 *  School of Computer Science at Chungbuk National University
 */

#include <stdio.h>
#include <stdlib.h>

#define MAX_QUEUE_SIZE 4

typedef char element;
typedef struct { // 구조체 선언
    element queue[MAX_QUEUE_SIZE];
    int front, rear;
}QueueType;


QueueType *createQueue();  // 함수 선언
int freeQueue(QueueType *cQ);
int isEmpty(QueueType *cQ);
int isFull(QueueType *cQ);
void enQueue(QueueType *cQ, element item);
void deQueue(QueueType *cQ, element* item);
void printQ(QueueType *cQ);
void debugQ(QueueType *cQ);
element getElement();

int main(void)
{
    QueueType *cQ = createQueue(); //QueueType 포인터 선언 후 createQueue 함수 값 할당
    element data; //element 형 변수 data 선언
    char command; // char 형 변수 command 선언

    printf("----- ohjaesik ------ 2022040014-----");
    do{ // Queue 연산을 위한 반복문
        printf("\n-----------------------------------------------------\n");
        printf("                   Circular Q                  \n");
        printf("-----------------------------------------------------\n");
        printf(" Insert=i,  Delete=d,   PrintQ=p,   Debug=b,   Quit=q \n");
        printf("-----------------------------------------------------\n");

        printf("Command = ");
        scanf(" %c", &command);

        switch(command) {
        case 'i': case 'I': //Queue 에 값을 대입
            data = getElement();
```

```
            enQueue(cQ, data);
            break;
        case 'd': case 'D': // Queue 에 front 값 삭제
            deQueue(cQ, &data);
            break;
        case 'p': case 'P': // Queue print
            printQ(cQ);
            break;
        case 'b': case 'B': // Queue debug
            debugQ(cQ);
            break;
        case 'q': case 'Q': // Queue 메모리 해제
            freeQueue(cQ);
            break;
        default: // 잘못 입력된 값 처리
            printf("\n       >>>>>   Concentration!!   <<<<<     \n");
            break;
        }

    }while(command != 'q' && command != 'Q'); // 종료


    return 1;
}

QueueType *createQueue()
{
    QueueType *cQ; // QueueType 포인터 변수 선언
    cQ = (QueueType *)malloc(sizeof(QueueType)); // cQ 가 가리키는 주소에 QueueType 의
크기의 메모리 할당
    cQ->front = 0; // cQ 가 가리키는 주소에 front 의 값 할당
    cQ->rear = 0; // cQ 가 가리키는 주소에 rear 의 값 할당
    return cQ;
}

int freeQueue(QueueType *cQ)
{
    if(cQ == NULL) return 1; // cQ 가 가르키는 주소에 할당된 메모리 해제
    free(cQ);
    return 1;
}

element getElement()
{
    element item;
    printf("Input element = ");
    scanf(" %c", &item); // 입력받은 값을 item 에 할당
    return item;
}
```

```c
/* complete the function */
int isEmpty(QueueType *cQ)
{
    if(cQ->front == cQ->rear){ // front 와 rear 의 값이 같으면 Queue 가 비어 있는 것임을 체크
        printf("Queuse is empty");
        return 0;
    }
    return 1;
}

/* complete the function */
int isFull(QueueType *cQ)
{
    if(cQ->front == (cQ->rear + 1)% MAX_QUEUE_SIZE){ //front 와 rear + 1 의 값이 같으면 Queue 가 꽉 차있는 것임을 체크
        printf("Queue is full");
        return 0;
    }
    return 1;
}


/* complete the function */
void enQueue(QueueType *cQ, element item)
{
    if(isFull(cQ)){
        cQ->queue[(++cQ->rear)%MAX_QUEUE_SIZE] = item; //queue 에 rear 가 가리키는 index 에 입력 받은 값을 저장
    }
}

/* complete the function */
void deQueue(QueueType *cQ, element *item)
{
    if(isEmpty(cQ)){
        cQ->queue[(++cQ->front)%MAX_QUEUE_SIZE] = 0; //queue 에 front 가 가리키는 index 에 저장된 값을 삭제
    }
}


void printQ(QueueType *cQ)
{
    int i, first, last;
```

```
    first = (cQ->front + 1)%MAX_QUEUE_SIZE; //front +1 이 MAX_QUEUE_SIZE 를 넘어
값을때 값을 줄여주기 위해 MAX_QUEUE_SIZE 로 나누어줌
    last = (cQ->rear + 1)%MAX_QUEUE_SIZE; //rear +1 이 MAX_QUEUE_SIZE 를 넘어 값을때
값을 줄여주기 위해 MAX_QUEUE_SIZE 로 나누어줌

    printf("Circular Queue : [");

    i = first;
    while(i != last){ //queue 의 first 값부터 last 값을 순서대로 출력
        printf("%3c", cQ->queue[i]);
        i = (i+1)%MAX_QUEUE_SIZE;

    }
    printf(" ]\n");
}


void debugQ(QueueType *cQ)
{

    printf("\n---DEBUG\n");
    for(int i = 0; i < MAX_QUEUE_SIZE; i++)
    {
        if(i == cQ->front) {
            printf("  [%d] = front\n", i); // queue 의 front 값을 찾아 출력
            continue;
        }
        printf("  [%d] = %c\n", i, cQ->queue[i]); // queue 의 모든 값을 출력

    }
    printf("front = %d, rear = %d\n", cQ->front, cQ->rear); // front 와 rear 출력
}
```

```
----- ohjaesik ------ 2022040014-----
------------------------------------------------------
                    Circular Q
------------------------------------------------------
 Insert=i,  Delete=d,   PrintQ=p,   Debug=b,   Quit=q
------------------------------------------------------
Command = i
Input element = 1

------------------------------------------------------
                    Circular Q
------------------------------------------------------
 Insert=i,  Delete=d,   PrintQ=p,   Debug=b,   Quit=q
------------------------------------------------------
Command = i
Input element = 2

------------------------------------------------------
                    Circular Q
------------------------------------------------------
 Insert=i,  Delete=d,   PrintQ=p,   Debug=b,   Quit=q
------------------------------------------------------
Command = i
Input element = 3

------------------------------------------------------
                    Circular Q
------------------------------------------------------
 Insert=i,  Delete=d,   PrintQ=p,   Debug=b,   Quit=q
------------------------------------------------------
Command = d

------------------------------------------------------
                    Circular Q
------------------------------------------------------
 Insert=i,  Delete=d,   PrintQ=p,   Debug=b,   Quit=q
------------------------------------------------------
Command = b

---DEBUG
  [0] =
  [1] = front
  [2] = 2
  [3] = 3
front = 1, rear = 3

------------------------------------------------------
                    Circular Q
------------------------------------------------------
 Insert=i,  Delete=d,   PrintQ=p,   Debug=b,   Quit=q
------------------------------------------------------
Command =
```

Postfix.c

```c
/**
 * postfix.c
 *
 * School of Computer Science,
 * Chungbuk National University
 */
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX_STACK_SIZE 10
#define MAX_EXPRESSION_SIZE 20

/* stack 내에서 우선순위는 내림차순, lparen = 0 가장 낮음 */
typedef enum{
    lparen = 0, /* ( 왼쪽 괄호 */
    rparen = 9, /* ) 오른쪽 괄호*/
    times = 7, /* * 곱셈 */
    divide = 6, /* / 나눗셈 */
    plus = 5, /* + 덧셈 */
    minus = 4, /* - 뺄셈 */
    operand = 1 /* 피연산자 */
} precedence;
char infixExp[MAX_EXPRESSION_SIZE];
char postfixExp[MAX_EXPRESSION_SIZE];
char postfixStack[MAX_STACK_SIZE];
int evalStack[MAX_STACK_SIZE];

int postfixStackTop = -1;
int evalStackTop = -1;
int evalResult = 0;

void postfixpush(char x); // 함수 선언
char postfixPop();
void evalPush(int x);
int evalPop();
void getInfix();
precedence getToken(char symbol);
precedence getPriority(char x);
void charCat(char* c);
void toPostfix();
void debug();
void reset();
void evaluation();
```

```c
int main()
{
    char command;

    printf("----- ohjaesik ------ 2022040014-----");
    do{ // 함수를 사용하기 위한 반복문
        printf("----------------------------------------------------------------------
\n");
        printf(" 훆 훆 훆 훆 훆 훆 훆 Infix to Postfix, then Evaluation 훆 훆 훆 훆 훆
훆 훆 \n");
        printf("----------------------------------------------------------------------
\n");
        printf(" Infix=i, 훆 Postfix=p, 훆 Eval=e, 훆 Debug=d, 훆 Reset=r, 훆 Quit=q
\n");
        printf("----------------------------------------------------------------------
\n");

        printf("Command = ");
        scanf(" %c", &command);

        switch(command) {
        case 'i': case 'I':
            getInfix();
            break;
        case 'p': case 'P':
            toPostfix();
            break;
        case 'e': case 'E':
            evaluation();
            break;
        case 'd': case 'D':
            debug();
            break;
        case 'r': case 'R':
            reset();
            break;
        case 'q': case 'Q':
            break;
        default:
            printf("\n 훆 훆 훆 >>>>> 훆 Concentration!! 훆 <<<<< 훆 훆 \n");
            break;
        }

    }while(command != 'q' && command != 'Q');

    return 1;
}

void postfixPush(char x) // postfixPush 구현
```

```
{
    postfixStack[++postfixStackTop] = x;
}

char postfixPop() // postfixPop 구현
{
 char x;
 if(postfixStackTop == -1)
  return '\0';
 else {
    x = postfixStack[postfixStackTop--];
 }
 return x;
}

void evalPush(int x) // evalPush 구현
{
evalStack[++evalStackTop] = x;
}

int evalPop() //evalPop 구현
{
if(evalStackTop == -1)
return -1;
else
return evalStack[evalStackTop--];
}

/**
 * infix expression을 입력받는다.
 * infixExp에는 입력된 값을 저장한다.
 */
void getInfix()
{
printf("Type the expression >>> ");
scanf("%s",infixExp);
}

precedence getToken(char symbol)
{
    switch(symbol) {
    case '(' : return lparen;
    case ')' : return rparen;
    case '+' : return plus;
    case '-' : return minus;
    case '/' : return divide;
    case '*' : return times;
    default : return operand;
    }
```

```
}

precedence getPriority(char x)
{
    return getToken(x);
}

/**
 * 문자하나를 전달받아, postfixExp 에 추가
 */
void charCat(char* c)
{
    if (postfixExp == '\0')
        strncpy(postfixExp, c, 1);
    else
        strncat(postfixExp, c, 1);
}

/**
 * infixExp 의 문자를 하나씩 읽어가면서 stack 을 이용하여 postfix 로 변경한다.
 * 변경된 postfix 는 postFixExp 에 저장된다.
 */
void toPostfix()
{
    /* infixExp 의 문자 하나씩을 읽기위한 포인터 */
    char *exp = infixExp;
    int idx = 0;
    /* exp 를 증가시켜가면서, 문자를 읽고 postfix 로 변경 */
    while(*exp != '\0'){ // exp 가 끝날 때까지 반복

        if(getPriority(*exp)== operand){  //exp 가 숫자일 경우 배열에 추가
            postfixExp[idx++] = *exp++;

        }
        else if( getPriority(*exp)==lparen) // exp 가 '('일 경우 배열에 우선적으로 추가
            postfixPush(*exp++);
        else if (getToken(*exp)==rparen){ // exp 가 ')'가 나왔을 경우 '('가 나올 때까지
기호를 팝함.
            while(getToken(postfixStack[postfixStackTop])!=lparen){
                postfixExp[idx++] = postfixPop();
            }
            postfixPop(); // '('을 팝.
            *exp++;
        }
        else if( postfixStackTop == -1 || getToken(postfixStack[postfixStackTop]) <
getToken(*exp)){ //우선 순위가 큰 기호 스택에 저장
            postfixPush(*exp++);
        }
```

```
        else {
            while(getToken(postfixStack[postfixStackTop]) >= getToken(*exp)){ //
우선 순위가 낮거나 같은 연산자들을 팝함.
            postfixExp[idx++] = postfixPop();
            }
            postfixPush(*exp++); // 이후 연산자 스택에 저장

        }
    }

    while(postfixStackTop != -1){ // 스택에 저장되어 있는 모든 연산자 출력
        postfixExp[idx++] = postfixPop();

    }


}
void debug()
{
    printf("\n---DEBUG\n");
    printf("infixExp = %s\n", infixExp);
    printf("postExp = %s\n", postfixExp);
    printf("eval result = %d\n", evalResult);

    printf("postfixStack : ");
    for(int i = 0; i < MAX_STACK_SIZE; i++)
        printf("%c ", postfixStack[i]);

    printf("\n");

}

void reset()
{
    infixExp[0] = '\0';
    postfixExp[0] = '\0';

    for(int i = 0; i < MAX_STACK_SIZE; i++)
        postfixStack[i] = '\0';
    postfixStackTop = -1;
    evalStackTop = -1;
    evalResult = 0;
}

void evaluation()
{
    int idx=0;
    int x;
    while(postfixExp[idx] != '\0'){ // postfix 의 끝까지 도는 반복문
        x=getToken(postfixExp[idx]); //x 에 postfixExp 배열의 token 값을 할당
```

```
        if(x==operand){ //숫자면 int 형으로 바꾸고 저장
            evalPush(postfixExp[idx++] - 48);
        }
        else{ // 연산자가 나왔을 시 연산자에 맞게 계산
            int intger2 = evalPop();
            int intger1 = evalPop();
            if(x==minus){
                evalPush(intger1 - intger2);
            }
            else if(x==times){
                evalPush(intger1 * intger2);
            }
            else if(x==divide){
                evalPush(intger1 / intger2);
            }
            else{
                evalPush(intger1 + intger2);
            }
            idx++;
        }
    }
    evalResult = evalPop(); // 스택에 첫번째값이 결과 값이므로 결과값을 evalResult 에
할당.
}
```

```
----- ohjaesik ------ 2022040014----------------------------------------------
 흙 흙 흙 흙 흙 흙 흙 Infix to Postfix, then Evaluation 흙 흙 흙 흙 흙 흙 흙
--------------------------------------------------------------------
 Infix=i, 흙 Postfix=p, 흙 Eval=e, 흙 Debug=d, 흙 Reset=r, 흙 Quit=q
--------------------------------------------------------------------
Command = i
Type the expression >>> 1*1*2+2*3
--------------------------------------------------------------------
 흙 흙 흙 흙 흙 흙 흙 Infix to Postfix, then Evaluation 흙 흙 흙 흙 흙 흙 흙
--------------------------------------------------------------------
 Infix=i, 흙 Postfix=p, 흙 Eval=e, 흙 Debug=d, 흙 Reset=r, 흙 Quit=q
--------------------------------------------------------------------
Command = p
--------------------------------------------------------------------
 흙 흙 흙 흙 흙 흙 흙 Infix to Postfix, then Evaluation 흙 흙 흙 흙 흙 흙 흙
--------------------------------------------------------------------
 Infix=i, 흙 Postfix=p, 흙 Eval=e, 흙 Debug=d, 흙 Reset=r, 흙 Quit=q
--------------------------------------------------------------------
Command = d

---DEBUG
infixExp = 1*1*2+2*3
postExp = 11*2*23*+
eval result = 0
postfixStack : + *
--------------------------------------------------------------------
 흙 흙 흙 흙 흙 흙 흙 Infix to Postfix, then Evaluation 흙 흙 흙 흙 흙 흙 흙
--------------------------------------------------------------------
 Infix=i, 흙 Postfix=p, 흙 Eval=e, 흙 Debug=d, 흙 Reset=r, 흙 Quit=q
--------------------------------------------------------------------
Command = e
--------------------------------------------------------------------
 흙 흙 흙 흙 흙 흙 흙 Infix to Postfix, then Evaluation 흙 흙 흙 흙 흙 흙 흙
--------------------------------------------------------------------
 Infix=i, 흙 Postfix=p, 흙 Eval=e, 흙 Debug=d, 흙 Reset=r, 흙 Quit=q
--------------------------------------------------------------------
Command = d

---DEBUG
infixExp = 1*1*2+2*3
postExp = 11*2*23*+
eval result = 8
postfixStack : + *
--------------------------------------------------------------------
 흙 흙 흙 흙 흙 흙 흙 Infix to Postfix, then Evaluation 흙 흙 흙 흙 흙 흙 흙
--------------------------------------------------------------------
 Infix=i, 흙 Postfix=p, 흙 Eval=e, 흙 Debug=d, 흙 Reset=r, 흙 Quit=q
--------------------------------------------------------------------
Command =
```

```
➜  자료구조 git:(main) ✗ git add circularQ.c
➜  자료구조 git:(main) ✗ git add postfix.c
➜  자료구조 git:(main) ✗ git commit -m '커밋 기록 남기기'
[main 47ecaa1] 커밋 기록 남기기
 Committer: 오재식 <ojaesig@ojaesigs-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 2 insertions(+)
➜  자료구조 git:(main) ✗ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 394 bytes | 394.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/ohjaesik/homework5.git
   2930ddf..47ecaa1  main -> main
```

https://github.com/ohjaesik/homework5